Computing Probable Maximum Loss in Catastrophe Reinsurance Portfolios on Multi-Core and Many-Core Architectures

Neil Burke, Andrew Rau-Chaplin
Faculty of Computer Science
Dalhousie University, Canada
Email: neil.burke@dal.ca, arc@cs.dal.ca

Blesson Varghese
School of Computer Science
University of St Andrews, UK
Email: varghese@st-andrews.ac.uk

***Abstract*—In the reinsurance market, the risks natural catastrophes pose to portfolios of properties must be quantified, so that they can be priced, and insurance offered. The analysis of such risks at a portfolio level requires a simulation of up to 800,000 trials with an average of 1,000 catastrophic events per trial. This is sufficient to capture risk for a global multi-peril reinsurance portfolio covering a range of perils including earthquake, hurricane, tornado, hail, severe thunderstorm, wind storm, storm surge and riverine flooding, and wildfire. Such simulations are both computation and data intensive, making the application of high-performance computing techniques desirable.**

In this paper we explore the design and implementation of Portfolio Risk Analysis (PRA) on both multi-core and many-core computing platforms. Given a portfolio of property catastrophe insurance treaties, key risk measures, such as Probable Maximum Loss (PML) are computed by taking both primary and secondary uncertainty into account. *Primary Uncertainty* is associated with whether or not an event occurs in a simulated year, while *Secondary Uncertainty*, captures the uncertainty in the level of loss due to the use of simplified physical models and limitations in the available data. A combination of fast lookup structures, multi-threading, and careful hand tuning of numerical operations is required to achieve good performance. Experimental results are reported for multi-core processors and systems using NVIDIA GPU and Intel Phi many-core accelerators.

Keywords*—*risk analysis; secondary uncertainty; many-core computing; hardware accelerators

I. Introduction

Reinsurance companies, who insure primary insurance companies against losses associated with natural catastrophes, such as earthquakes, hurricanes and floods, must quantify the risk associated with large portfolios of risk transfer treaties. Given a portfolio of complex property catastrophe insurance contracts (including Cat eXcess of Loss (XL), Per-Occurrence XL, and Aggregate XL treaties), Portfolio Risk Analysis (PRA) [1]–[3] is performed to compute risk measures including Probable Maximum Loss (PML) [4] and the Tail Value-at-Risk (TVaR) [5]. PRA is central to both treaty pricing and portfolio/solvency applications, but is very computationally intensive as it may involve 1,000,000 trial simulations in which each trial consists of 800 to 1200 simulated catastrophic events, each of which may impact 10,000 to 1,000,000 individual buildings (or locations). Given the data and computational intensity of this analysis the application of High Performance Computing (HPC) techniques is of significant interest to the industry.

Previously the design and implementation of parallel methods for PRA was explored [6]. However the analysis was of limited use for portfolio wide risk analysis scenarios since it only accounted for Primary Uncertainty - the uncertainty associated with whether a catastrophic event occurs or not in a simulated year. Secondary Uncertainty, the uncertainty in the amount of loss given that the event occurs also needs to be considered. Essentially, this is the difference between a simulation through which mean loss values flow and a simulation in which loss distributions flow.

In practice there are many sources of secondary uncertainty in catastrophic risk modelling. For example, the exposure data which describes the buildings, their locations, and construction types may be incomplete, lacking in sufficient detail, or simply inaccurate. Also the physical modelling of the hazard, for example an earthquake, may naturally generate a distribution of hazard intensity values due to uncertainty associated with the energy attenuation functions used for driving data such as soil type. Lastly, building vulnerability functions are simplifications of complex physical phenomenon and are therefore much better at producing loss distributions than accurate point estimates. A model that accounts only for primary uncertainty uses only mean loss values and fails to account for what is known about the loss distribution. However, a model that can account for both primary and secondary uncertainty can accept as input complete event loss distributions represented by the event rate, mean loss, independent standard deviation, and correlated standard deviation, and better account for the range of possible outcomes.

In this paper, a Portfolio Risk Analysis engine capable of capturing both primary and secondary uncertainty is presented. The engine is designed to run on multi-core systems such as Intel Xeon and i7 and many-core systems including NVIDIA GPUs and Intel Xeon Phi. In order to account for secondary uncertainty, distributions of loss values are carried through the analysis, rather than simple mean values. Performing computations on these distributions requires efficient statistical operations such as quantiles of the Beta distribution. This poses a significant challenge to parallelisation because rather than balancing work across threads performing fixed time operations, like simple addition, we must now balance the work in the face of individual numerical operations whose run times are highly variable since, for example, they are based on iterative methods. Additionally, for efficient parallelisation we need to adapt the workload to hardware architectures by making use of SIMD vector registers, but the variable nature of the number of computational steps in iterative methods for obtaining quantiles of the Beta distribution makes this difficult.

This paper describes the methodology, the implementation, and optimisation for many-core architectures, and reports an experimental evaluation. Since one of the keys to the performance on these

algorithms was the efficiency of the underlying statistical operations, a number of different statistical libraries and hand coded numeric codes were explored. For benchmarking purposes, we used a PML computation requiring a parallel simulation of 800,000 trials with 1,000 catastrophic events per trial on an exposure set, and on a multi-layered contract structure taking secondary uncertainty into account. These parameters were selected in conjunction with industry practitioners to capture realistic analysis problems. On the multi-core CPUs, a combination of fast lookup structures, multi-threading, and careful hand tuning lead to an optimal linear speed up. A speed up of 18x was achieved on a 16-core Xeon system and 4.3x on a 4-core i7, the slight super linear effects observed due to hyperthreading. The complete computation of PML with secondary uncertainty for a complex multi-layered reinsurance contract covering global region/peril exposure can now be executed in under 140 seconds on a i7 workstation and under 50 seconds on a dual-socket Xeon server.

In addition, simulation codes were developed for the NVIDIA Tesla GPUs and Intel Phi Accelerators. These codes run largely in offload mode with the host CPUs responsible for preprocessing and data transfer and the accelerators performing all numerical calculations. Total runtime on the GPU was 56 seconds making it 2.5x faster than the i7 runtime and 0.87x of the performance of our high-end two socket Xeon server. Note that a hybrid version of the simulation that harnessed both the Xeon and GPU simultaneously for the numerical calculations would be faster still. Before we started the design and implementation of our Portfolio Risk Analysis engine on the accelerator cards, we had expected that the Intel Phi would outperform the NVIDIA GPU in two distinct ways. Firstly, we expected development effort to be much reduced on the Phi because of the shared software stack and the ability to reuse much of the Intel multi-core implementation. Secondly, we expected the Intel Phi to outperform our older NVIDIA GPU platform because of its higher peak FLOPS and memory bandwidth performance. While development effort was indeed considerably less on the Phi the achieved performance was significantly lower than on the GPU. Total runtime on the Phi, including all data transfer from host to accelerator was 89 seconds, making it 1.6x faster than the i7 runtime and 0.55x of the performance of our high-end two socket Xeon server. Careful profiling of the different versions of our PRA engine reveals that the cost of computation is dominated by two operations: random access memory lookups (for which caching is of little help), and computation of the inverse beta cumulative distribution function (which is based on an iterative algorithm that resists vectorization). These key operations are difficult to efficiently parallelise in general but are more robustly handled by the GPU allowing our GPU-based PRA engine to achieve a higher percentage of the hardware's peak performance.

The remainder of this paper is organised as follows. Section II presents a discussion on related work. Section III considers the inputs, methodology and output of portfolio risk analysis as well as computing secondary uncertainty. Section IV presents the implementation of the analysis on two many-core hardware architectures. Section V show highlights the key results obtained from the experimental evaluation. Section VI concludes this paper.

II. Related Work

Parallelism has been widely used in the domain of computational finance and risk [7]–[9]. Financial and risk applications in the production setting have progressed from small scale clusters [10], [11] to large supercomputers [12], [13]. Recently, an increasing number of financial applications have been migrating from small clusters to multi-core processors and many-core accelerators which are available at a low cost budget [14]. This includes use of Cell BE processors [15], [16] and custom FPGAs architectures [17], [18]. Also, independent GPU acceleration is now frequently employed [19], [20]. Use of heterogeneous clusters which mix CPU and GPU processors, is another popular platform [21], [22].

Although high-performance computing platforms are an option to accommodate and accelerate risk simulations, there is an investment cost that will need to be borne along with maintenance costs. A relatively cost effective solution is to employ hardware accelerators to address the problems faced by current risk simulations - (i) hardware accelerators can provide fast numerical computations which are required by statistical functions that support applying secondary uncertainty in PRA, (ii) hardware accelerators can provide a platform to facilitate ad hoc risk simulations because they are not only cost effective but can address the memory and computational challenges of the simulations, and (iii) hardware accelerators can be exploited to achieve speed up and thereby use risk simulations in real-time.

Much of the research into the application of high-performance computing techniques to simulations in the area of risk, specifically the insurance and reinsurance domain, has been conducted in an industrial setting and not published in the scientific literature for competitive reasons. However, this situation is slowly changing with recent papers on parallel catastrophe modelling [23], risk treaty optimization [24], [25] and parallel reinsurance risk analytics [26]–[28].

III. Portfolio Risk Analysis

Monte Carlo simulations are required for portfolio risk management and contract pricing. Such a simulation in which each trial of the simulation represents a distinct view of which events occur and in what order they occur in a contractual year is referred to as Portfolio Risk Analysis (PRA). One merit of performing such an analysis is that millions of alternative views of a single contractual year can be obtained. This simulation is based on a pre-generated Year Event Table (YET). This section will consider the inputs required for performing PRA, propose a methodology for aggregating losses by taking primary and secondary uncertainty in account, consider the financial terms employed in the algorithms, and present the output of the analysis.

A. Inputs

Three inputs are required to perform PRA; the first is a large table consisting of nearly one million trials, the second is a set of modelled single event losses, and the third is metadata describing the portfolio of covering financial treaties.

1) Year Event Table: denoted as YET, contains the occurrence and timing of catastrophic events for a year. The YET is generated by specialists in the underlying hazards such as seismologists, meteorologist, and hydrologists, for earthquake, hurricane and flood events respectively. The YET provides a million distinct views of potential events that can occur in a year.

Each record in a YET is called a Trial, denoted as T_i, which represents a possible sequence of event occurrences for any given year. The sequence of events is defined by an ordered set of tuples containing the ID of an event and the time-stamp of its occurrence in that trial

$$T_i \quad = \quad \{(E_{i,1}, t_{i,1}, z_{(Prog,E)_{i,1}}), \ldots, (E_{i,k}, t_{i,k}, z_{(Prog,E)_{i,k}})\}.$$

The set is ordered by ascending time-stamp values. Program-and-Event-Occurrence-Specific random number, $z_{(Prog,E)}$ is further considered later in this section. A typical YET may comprise thousands to millions of trials, and each trial may have approximately 800 to 1500 'event time-stamp' pairs, based on a global event catalogue covering multiple perils. The YET can be represented as

$$YET \quad = \quad \{T_i = \{(E_{i,1}, t_{i,1}, z_{(Prog,E)_{i,1}}), \ldots, (E_{i,k}, t_{i,k}, z_{(Prog,E)_{i,k}})\}\},$$

where $i = 1, 2, \ldots$ ***and*** $k = 1, 2, \ldots, 1500$.

2) Extended Event Loss Tables: ***The frequency, physical characteristics, and impact of the potential events are estimated by a Catastrophe Model [3] which usually consists of hazard, vulnerability and loss modules. The output of a catastrophe model is denoted as*** $XELT$***, which represents a collection of specific events and their corresponding losses with respect to an exposure set. In addition, a few parameters, namely the Event-Occurrence-Specific random number (*** $z_{(E)}$***), the independent standard deviation of loss (*** σ_I***), the correlated standard deviation of loss (*** σ_C***), and the maximum expected loss (*** max_l***). The loss associated with an event*** E_i ***is represented as*** μ_l ***and is required for analysis with secondary uncertainty. Applying secondary uncertainty using the XELT is presented later in this section.***

Each record in an XELT is denoted as 'eXtended' event loss

$$XEL_i \quad = \quad \{E_i, l_i, z_{(E)_i}, \sigma_{I_i}, \sigma_{C_i}, max_{l_i}\}.$$

and the financial terms associated with the XELT are represented as a tuple

$$\mathcal{I} \quad = \quad (\mathcal{I}_1, \mathcal{I}_2, \ldots).$$

A typical aggregate analysis may comprise 10,000 XELTs, each containing 10,000 to 30,000 extended event losses with exceptions even up to 2,000,000 extended event losses. The XELTs can be represented as

$$XELT = \left\{ \begin{array}{lcl} XEL_i & = & \{E_i, \mu_{l_i}, z_{(E)_i}, \\ & & \sigma_{I_i}, \sigma_{C_i}, max_{l_i}\}, \\ \mathcal{I} & = & (\mathcal{I}_1, \mathcal{I}_2, \ldots) \end{array} \right\}$$

with $i = 1, 2, \ldots, 30,000$.

3) Portfolio: ***is denoted as*** PF***, which contains a group of Programs, denoted as*** P ***and represented as***

$$PF \quad = \quad \{P_1, P_2, \cdots, P_n\}$$

with $n = 1, 2, \ldots, 10$.

Each Program in turn covers a set of Layers, denoted as L***, which cover a collection of XELTs under a set of layer terms. A single layer*** L_i ***is composed of two attributes. Firstly, the set of XELTs***

$$\mathcal{E} = \{XELT_1, XELT_2, \ldots, XELT_j\},$$

and secondly the Layer Terms, denoted as

$$\mathcal{T} = (\mathcal{T}_{OccR}, \mathcal{T}_{OccL}, \mathcal{T}_{AggR}, \mathcal{T}_{AggL}).$$

A typical Layer covers approximately 3 to 30 individual XELTs. The Layer can be represented as

$$L = \left\{ \begin{array}{lcl} \mathcal{E} & = & \{XELT_1, XELT_2, \ldots, XELT_j\}, \\ \mathcal{T} & = & (\mathcal{T}_{OccR}, \mathcal{T}_{OccL}, \mathcal{T}_{AggR}, \mathcal{T}_{AggL}) \end{array} \right\}$$

with $j = 1, 2, \ldots, 30$.

B. Methodology

The methodology (line no. 1-17 shown in Algorithm 1) for PRA has two stages. In the first stage, data is loaded into local memory which is referred to as the preprocessing stage in this paper. In this stage YET***,*** $XELT$***, and*** PF ***are loaded into memory.***

In the second stage, the four step simulation executed for each Layer and for each trial in the YET is performed as shown below and the resulting Year Loss Table (YLT) ***is produced.***

Algorithm 1: Aggregate Risk Analysis with Primary and Secondary Uncertainty

Input : YET, $XELT$, PF
Output : YLT

```
1  for each Program, P, in PF do
2      for each Layer, L, in P do
3          for each Trial, T, in YET do
4              for each Event, E, in T do
5                  for each XELT covered by L do
6                      Lookup E in the XELT and find
                       corresponding loss, l_E
7                      Apply Secondary Uncertainty to l_E
8                      Apply Financial Terms to l_E
9                      l_T ← l_T + l_E
10                 end
11                 Apply Occurrence Financial Terms to l_T
12                 Apply Aggregate Financial Terms to l_T
13             end
14         end
15     end
16 end
17 Populate YLT using l_T
```

In the first step (shown in line no. 6) each event of a trial and its corresponding event loss in the set of XELTs associated with the Layer is determined. In the second step shown in line nos. 7-9, secondary uncertainty is applied to each loss value of the Event-Loss pair extracted from an XELT. A set of contractual financial terms are then applied to the layer. For this the losses for a specific event's net of financial terms $\mathcal{I}$ ***are accumulated across all XELTs into a single event loss shown in line no. 9. In the third step in line no. 11 the event loss for each event occurrence in the trial, combined across all XELTs associated with the layer, is subject to occurrence terms. In the fourth step in line no. 12 aggregate terms are applied. The next sub-section will describe how the financial terms are applied.***

C. Applying Financial Terms

The financial terms applied on the loss values combined across all XELTs associated with the layer are Occurrence and Aggregate terms. Two occurrence terms, namely (i) Occurrence Retention, denoted as $\mathcal{T}_{OccR}$***, which is the retention or deductible of the insured for an individual occurrence loss, and (ii) Occurrence Limit, denoted as*** $\mathcal{T}_{OccL}$***, which is the limit of coverage the insurer will pay for occurrence losses in excess of the retention are applied. Occurrence terms are applicable to individual event occurrences independent of any other occurrences in the trial. The occurrence terms capture specific contractual properties of 'eXcess of Loss' treaties as they apply to individual event occurrences only. The event losses net of occurrence terms are then accumulated into a single aggregate loss for the given trial. The occurrence terms are applied as*** $l_T = min(max(l_T - \mathcal{T}_{OccR}), \mathcal{T}_{OccL})$.

Two aggregate terms, namely (i) Aggregate Retention, denoted as $\mathcal{T}_{AggR}$***, which is the retention or deductible of the insured for an annual cumulative loss, and (ii) Aggregate Limit, denoted as*** $\mathcal{T}_{AggL}$***, which is the limit or coverage the insurer will pay for annual cumulative losses in excess of the aggregate retention are applied. Aggregate terms are applied to the trial's aggregate loss for a layer. Unlike occurrence terms, aggregate terms are applied to the cumulative sum of occurrence losses within a trial and thus the result depends on the sequence of prior events in the trial. This***

behaviour captures contractual properties as they apply to multiple event occurrences. The aggregate loss net of the aggregate terms is referred to as the trial loss or the year loss. The aggregate terms are applied as $l_T = min(max(l_T - \mathcal{T}_{AggR}), \mathcal{T}_{AggL})$.

D. Output

The output of the algorithm for performing aggregate risk analysis with primary and secondary uncertainty is a loss value associated with each trial of the YET. A reinsurer can derive important portfolio risk metrics such as the Probable Maximum Loss (PML) and the Tail Value-at-Risk (TVaR) which are used for both internal risk management, and reporting to regulators and rating agencies. Furthermore, these metrics flow into a final stage of the risk analytics pipeline, namely Enterprise Risk Management, where liability, asset, and other forms of risks are combined and correlated to generate an enterprise wide view of risk.

Additional functions can be used to generate reports that will aid actuaries and decision makers. For example, reports presenting Return Period Losses (RPL) by Line of Business (LOB), Class of Business (COB) or Type of Participation (TOP), Region/Peril losses, Multi-Marginal Analysis and Stochastic Exceedance Probability (STEP) Analysis.

E. Secondary Uncertainty

The methodology to compute secondary uncertainty draws on industry-wide practices. The inputs required for the secondary uncertainty method and the sequence of steps for applying uncertainty to estimate a loss are considered in this section.

The following six inputs are required for computing secondary uncertainty, and are obtained from the Year Event Table (YET) and the 'eXtended ELT' (XELT):

i. $z_{(Prog,E)} = P_{(Prog,E)} \in U(0,1)$ *referred to as the Program-and-Event-Occurrence-Specific random number. Each event occurrence across different Programs have different random numbers.*
ii. $z_{(E)} = P_{(E)} \in U(0,1)$ *referred to as the Event-Occurrence-Specific random number. Each Event occurrence across different Programs have the same random number.*
iii. μ_l *referred to as the mean loss.*
iv. σ_I *referred to as the independent standard deviation of loss and represents the variance within the event-loss distribution.*
v. σ_C *referred to as the correlated standard deviation of loss and represents the error of the event-occurrence dependencies.*
vi. max_l *referred to as the maximum expected loss.*

Given the above inputs, the independent and correlated standard deviations need to be combined to reduce the error in estimating the loss value associated with an event. This is done in the following five steps:

i. *The raw standard deviation is produced as* $\sigma = \sigma_I + \sigma_C$.
ii. *The probabilities of occurrences,* $z_{(Prog,E)}$ *and* $z_{(E)}$ *are transformed from uniform distribution to normal distribution using*

$$f(x;\mu,\sigma^2) = \int_{-\infty}^{x} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \mathrm{d}x$$

This is applied to the probabilities of event occurrences as

$$\begin{aligned} v_{(Prog,E)} &= f(z_{(Prog,E)};0,1) \in N(0,1) \\ v_{(E)} &= f(z_{(E)};0,1) \in N(0,1) \end{aligned}$$

iii. *The linear combination of the transformed probabilities of event occurrences and the standard deviations is computed as*

$$LC = v_{(Prog,E)}\left(\frac{\sigma_I}{\sigma}\right) + v_{(E)}\left(\frac{\sigma_C}{\sigma}\right)$$

iv. *The normal random variable is computed as*

$$v = \frac{LC}{\sqrt{\left(\frac{\sigma_I}{\sigma}\right)^2 + \left(\frac{\sigma_C}{\sigma}\right)^2}}$$

v. *The normal random variable is transformed from normal distribution to uniform distribution as*

$$z = \Phi(v) = F_{Norm}(v) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{v} e^{\frac{-t^2}{2}} \mathrm{d}t$$

The model used above for combining the independent and correlated standard deviations represents two extreme cases. The first case in which $\sigma_I = 0$ *and the second case in which* $\sigma_C = 0$. *The model also ensures that the final random number,* z, *is based on both the independent and correlated standard deviations.*

The loss is estimated using the Beta distribution since fitting such a distribution allows the representation of risks quite accurately. The Beta distribution is a two parameter distribution, with an upper bound for the standard deviation. The standard deviation, mean, Alpha and Beta are defined as

$$\begin{aligned} \sigma_\beta &= \frac{\sigma}{max_l} \\ \mu_\beta &= \frac{\mu_l}{max_l} \\ \alpha &= \mu_\beta\left(\left(\frac{\sigma_{\beta_{max}}}{\sigma_\beta}\right)^2 - 1\right) \\ \beta &= (1-\mu_\beta)\left(\left(\frac{\sigma_{\beta_{max}}}{\sigma_\beta}\right)^2 - 1\right) \end{aligned}$$

An upper bound is set to limit the standard deviation using $\sigma_{\beta_{max}} = \sqrt{\mu_\beta(1-\mu_\beta)}$, *if* $\sigma_\beta > \sigma_{\beta_{max}}$, *then* $\sigma_\beta = \sigma_{\beta_{max}}$. *In the algorithm reported in this paper, for numerical purposes a value very close to* $\sigma_{\beta_{max}}$ *is chosen.*

To obtain the loss after applying secondary uncertainty beta distribution functions are used as follows

$$\begin{aligned} Loss &= max_l * InvCDF_{beta}(z;\alpha,\beta) \\ InvCDF_{beta}(z;\alpha,\beta) &= \left(\frac{B(z;\alpha,\beta)}{B(\alpha,\beta)}\right)^{-1}, \text{ where} \\ B(z;\alpha,\beta) &= \int_0^z t^{\alpha-1}(1-t)^{\beta-1}\mathrm{d}t \end{aligned}$$

IV. Implementation

In this section, the hardware platforms employed for implementing PRA, the statistical libraries used for computing secondary uncertainty and the optimizations on the platforms are considered.

A. Hardware Platforms

We used two multi-core systems. The first consisted of a 3.40 GHz quad core Intel(R) Core (TM) i7-2600 processor with 16.0 GB of RAM, 256 KB L2 cache per core, 8MB L3 cache, and a maximum memory bandwidth of 21 GB/sec. The second consisted of a dual socket server with 2 Intel Xeon E5-2650 Sandy Bridge 2.00 GHz Eight Core processors with 20MB L3 Cache, DDR3-1600, 8.00GT/sec QPI.

The i7 system also acted as the host for the following two hardware accelerator platforms. The first is an NVIDIA Tesla C2075 GPU, consisting of 448 CUDA cores, each with a frequency of 1.15

GHz, a global memory of 6 GB and a memory bandwidth of 144 GB/sec. The peak double precision floating point performance is nearly 0.515 Tflops. The GPU implementation of PRA is compiled using the NVIDIA CUDA Compiler (nvcc), version 5.0 [29] and '`-arch sm_13`'.

The second is an Intel Xeon Phi Coprocessors 5110P consisting of 60 cores with a frequency of 1.053 GHz. The coprocessor supports a maximum of 240 threads, 8 GB od memory, and a memory bandwith of 320 GB/sec. The peak double precision floating point performance is close to one Tflop. The Phi Coprocessor is based on the Intel (R) Many Integrate Cores (MIC) architecture and supports the Intel Initial Many-Core Instructions (IMCI). The host has two 2.00 GHz octa core Intel (R) Xeon(R) E5-2650 processors with 256 GB of RAM. The processor has a 20 MB cache and a maximum memory bandwidth of 51.2 GB/sec.

The Intel processors supports Advanced Vector Extensions (AVX) instructions for vector operations. C++ and OpenMP are employed to exploit parallelism on the CPU and compiled using the icpc compiler, version 13.1 provided by the Intel(R) Compiler Suite. The '`-O3`' compiler flag was used for optimisation, the '`-openmp`' was used to include the OpenMP directive and the '`-mkl`' flag to include the Intel Math Kernel library.

The GPU and the Phi are employed in offload mode, which refers to the host supporting the data preprocessing and transfer activities and the accelerator performs all the numerical computation. This facilitates side-by-side processing on the host and the device.

B. Multicore PRA

The losses for events in a trial need to be determined by looking up loss values in the XELT. The key design question is whether the data structure containing the event-loss pairs for all trials can be stored as a simple array and used as a direct access table or if a more compact representation, one that avoids storing zero entries, is required. While fast lookups can be obtained with a simple array representation, this performance is achieved at the cost of high memory usage. Owing to the relatively large amount of memory on the Xeon and i7 machines, the multi-core PRA implementation stores the XELT as a simple array.

C. GPU PRA

For the GPU implementation, all data structures are stored on the i7 host. Then a CUDA kernel is executed and the data is copied from the host to the device memory where all computations, including secondary uncertainty, for PRA are performed. The time taken to copy data is less than two seconds and therefore an alternative in which the host and the device operate asynchronously is not implemented. Similar to the multicore PRA a simple array is used to represent the XELT; compact representations are cumbersome to implement on complex memory hierarchies such as on GPUs. Each XELT is considered as an independent table. So in a read cycle, each thread independently looks up its events from the XELTs. All threads within a block access the same XELT.

D. Phi PRA

For the Intel Phi offloaded implementation, all data structures are initially stored in memory on the Xeon host in the same manner as the multi-core implementation. When analysis starts, the Xeon host begins by retrieving all data required for analysis on a chunk of trials and sends it to the Intel Phi. Once the Intel Phi receives the data, the coprocessor needs only to scan the buffer once in order to begin analysis on that chunk; no other data structure needs to be stored. While the Intel Phi analyses the input buffer, the Xeon host prepares the next buffer to send to the Phi in parallel, eliminating much of the cost of random access XELT lookups encountered in the multi-core implementation.

E. Libraries for Computing Secondary Uncertainty

Three statistical functions are required in the method for applying secondary uncertainty. They are (i) the Cumulative Distribution Function (CDF) of Normal distribution, (ii) the Quantile of the CDF of Normal distribution, and (iii) the Quantile of the Beta distribution. The Quantile of the Beta distribution is a numerically intensive function since an iterative method for achieving convergence of the solution within a certain error bound is generally employed.

Four statistical libraries are used to implement the statistical functions required for applying secondary uncertainty. Firstly, the Intel MKL library is used in the implementations on the CPU and Phi for the CDF of the Normal distribution and the Quantile of the Normal distribution [30]. The Intel MKL API currently does not support Beta distribution functions.

Secondly, the CUDA Math library is employed [31]. The CDF of the Normal distribution and the Quantile of the Normal Distribution are fast methods and included in the implementation. The CUDA Math API currently does not support Beta distribution functions.

Thirdly, the Boost statistical library is offered by the Boost C++ libraries [32] and is also used for the inverse beta distribution functions [33]. However, Boost is currently not supported for the GPU platform.

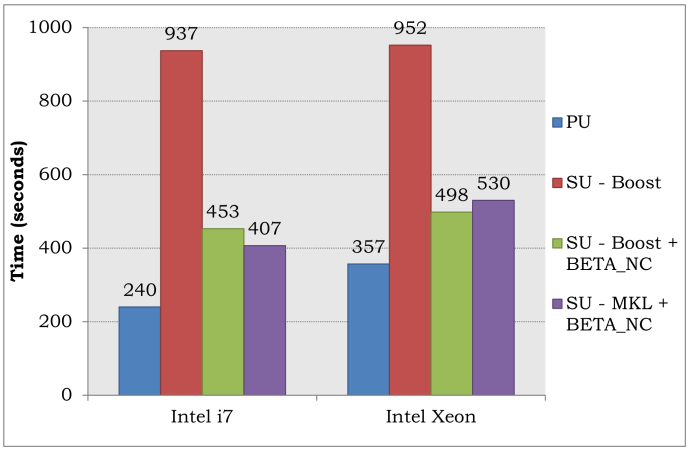
Fourthly, BETA_NC another C++ library that can evaluate the CDF of the Noncentral Beta distribution is employed [34]. This library was ported onto the GPU and Phi platforms.

V. Experimental Results

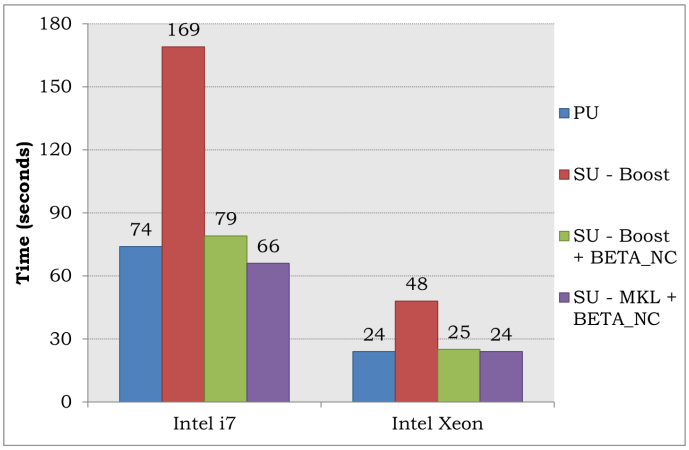
For benchmarking purposes, we used PML computations requiring a parallel simulation of 800,000 trials with an average of 1,000 catastrophic events per trial. This is sufficient to capture risk for a global multi-peril reinsurance portfolio covering earthquake, hurricane, tornado, hail, severe thunderstorm, wind storm, storm surge and riverine flooding, and wildfire. The PML analysis was conducted on multi-layered contract structures taking secondary uncertainty into account, where layers covered 15 XELTs on average. These parameters were selected in collaboration with industry practitioners to capture complex realistic analysis problems typical of the hardest cases that would be found in practice. In the figures below, we refer to Primary Uncertainty as PU and Secondary Uncertainty as SU. When two libraries, for example, Boost and BETA_NC are used (shown in the graphs as Boost + BETA_NC), the former is used for the statistical functions of primary uncertainty and the latter for secondary uncertainty.

Figure 1 shows the sequential and parallel execution times of PRA on multi-core architectures. The lowest times are obtained when the BETA_NC library is used for secondary uncertainty. Our multi-threaded PRA implementation runs in 140 seconds on the 4-core i7 (a 4.3x speedup over single threaded code) and 49 seconds in the 16-core Xeon server (a 17.9x speedup over single threaded code). The slightly superlinear speedups observed are due to hyperthreading.

Figure 2 shows the execution of PRA on the GPU. The number of threads per block are varied between 16 and 512, and the best performance is observed when 128 threads per block are employed. Beyond 128 threads per block the shared memory of the GPU is exhausted and an overhead is introduced for larger blocks. Total runtime on the GPU was 56 seconds, making it 2.5x faster than the i7 runtime and 0.87x of the performance of our high-end two



(a) Sequential



(b) Parallel (32 threads)

Fig. 1: PRA on multi-core architectures

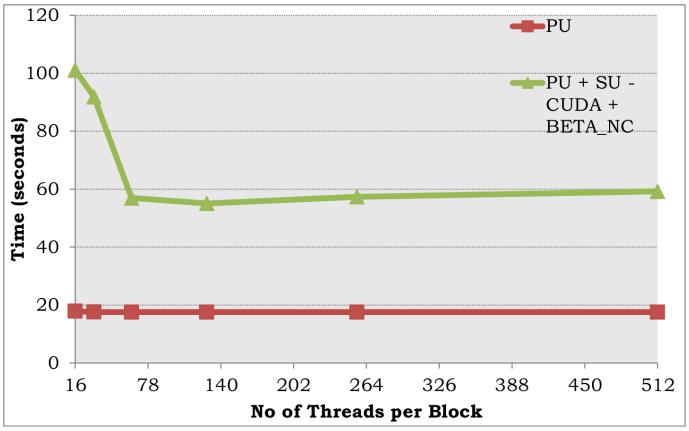


Fig. 2: PRA on the GPU

socket Xeon server. Note that a hybrid version of the PRA that uses both the Xeon and GPU simultaneously would be even faster.

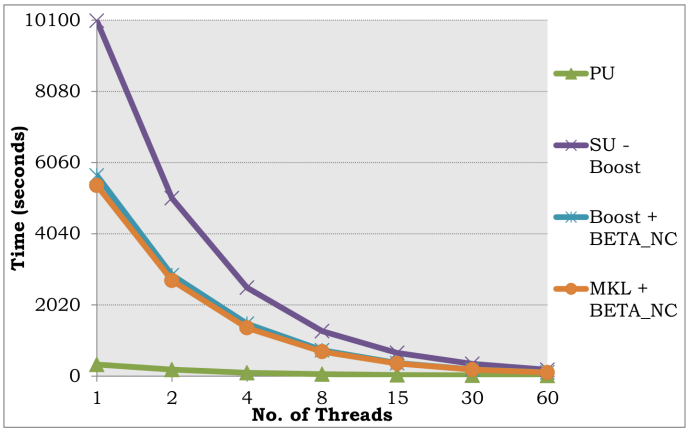
In Figure 3, the poor performance of Boost's Beta quantile function relative to the BETA_NC library's implementation is apparent. Since the normal statistical functions are very inexpensive compared to the Beta functions, changing the library for these functions had little effect on runtime. The Intel Phi has 60 physical cores, each of which can support up to four threads. In Figure 3b, we observe a 29% benefit when moving from 1 thread per core to 2 threads per core, and an additional 13% benefit when moving from 2 to 4 threads per core. The total runtime of PRA on the Phi, including all data transfer from host to accelerator, was 89 seconds making it 1.6x faster than the i7 runtime and 0.55x of the performance of our high-end two socket Xeon server. These runtime as somewhat disappointing, especially since the host is performing the loss lookups. We appear to be achieving a lower percentage of the accelerators peak hardware performance for two distinct reasons. Firstly, the wide vector units on the Phi are not well suited to executing iterative numerical operations like the cumulative inverse beta. Secondly, in trying to reduce development costs by reusing much of the Xeon code base, a Phi software ecosystem advantage much touted by Intel, the code is subjected to overheads in data transfers and time to reconstruct data structure on the Phi that might not be present in code that was developed from scratch.

Figure 4 summarises the time taken for applying financial terms, data transfer and memory look-ups, and computing secondary uncertainty. On both the Xeon and i7 platforms we achieve linear speed. The GPU version of our PRA engine demonstrates an excellent price/performance achieving 87% of the performance of a server costing four times as much. Our PRA implementation on the Phi is unable to take good advantage of the Phi's high peak performance numbers for reasons previously described. However, with significant further engineering we expect the performance gap between the two many-core accelerators could be narrowed.

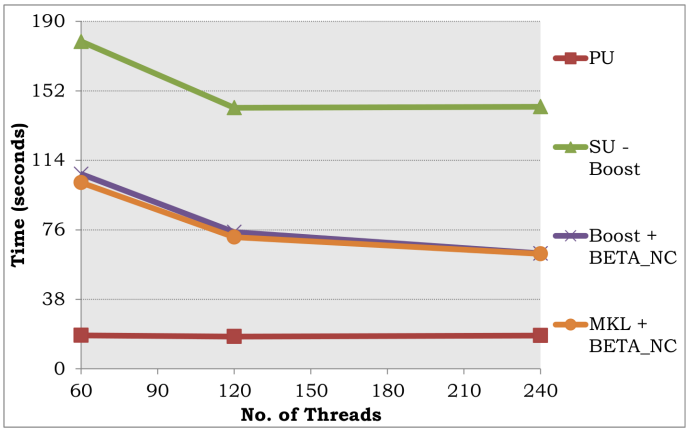
One obvious optimisation to improve the results on the Phi would be to better vectorize the BETA_NC library functions. In all the experiments presented in this paper the solution for the inverse Beta CDF converges when there is accuracy of up to six decimal places (or relative error is less than 10^{-6}). In our experiments, vectorising these functions at this level of relative error degraded their performance by about 25%. However, if the precision of the solution was increased gaining greater accuracy, for example, to twelve decimal places, then the vectorised library improved the performance significantly. When the precision is low only a few iterations are required for the solution to converge; a few iterations cannot reap the benefit of vectorisation but introduces overheads which is a trade-off.

VI. Conclusions

In this paper, we have explored the design and implementation of a Portfolio Risk Analysis (PRA) system on both multi-core and many-core computing platforms. We have shown how to compute, given a portfolio of property catastrophe insurance treaties, key risk measures such as Probable Maximum Loss (PML) while taking both primary and secondary uncertainty into account. With a combination of fast lookup structures, multi-threading, and careful hand tuning of numerical operations optimal linear speed up can be achieved on the multi-core platforms. Also, given substantial development effort, high performance can be obtained using GPU accelerators. Based on these approaches, it is now feasible on relatively low cost hardware platforms to perform Reinsurance Portfolio Risk Analysis in near real-time.



(a) One thread per core



(b) Multiple threads per core

Fig. 3: PRA on the Phi

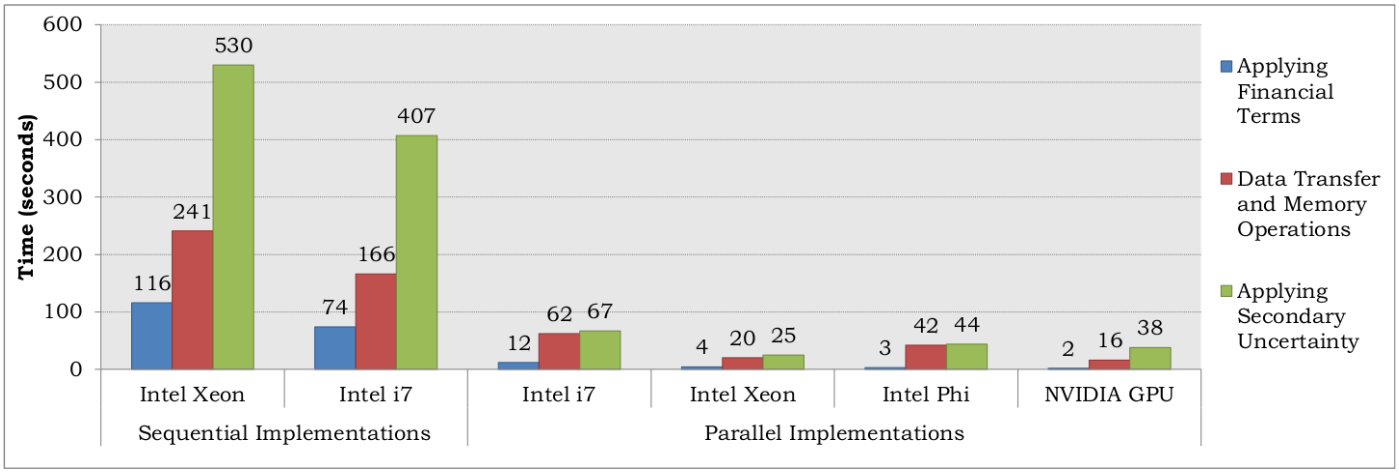


Fig. 4: Summary of best times of running PRA on multi-core and many-core platforms

REFERENCES

[1] *R. R. Anderson and W. Dong, "Pricing Catastrophe Reinsurance with Reinstatement Provisions using a Catastrophe Model," Casualty Actuarial Society Forum, Summer 1998, pp. 303-322.*

[2] *G. G. Meyers, F. L. Klinker and D. A. Lalonde, "The Aggregation and Correlation of Reinsurance Exposure," Casualty Actuarial Society Forum, Spring 2003, pp. 69-152.*

[3] *W. Dong, H. Shah and F. Wong, "A Rational Approach to Pricing of Catastrophe Insurance," Journal of Risk and Uncertainty, Vol. 12, 1996, pp. 201-218.*

[4] *G. Woo, "Natural Catastrophe Probable Maximum Loss," British Actuarial Journal, Vol. 8, 2002.*

[5] *A. A. Gaivoronski and G. Pflug, "Value-at-Risk in Portfolio Optimization: Properties and Computational Approach," Journal of Risk, Vol. 7, No. 2, 2005, pp. 1-31.*

[6] *A. K. Bahl, O. Baltzer, A. Rau-Chaplin, and B. Varghese, "Parallel Simulations for Analysing Portfolios of Catastrophic Event Risk," in Workshop Proceedings of the International Conference of High Performance Computing, Networking, Storage and Analysis (SC12), 2012.*

[7] *M. B. Giles and C. Reisinger, "Stochastic Finite Differences and Multilevel Monte Carlo for a Class of SPDEs in Finance," SIAM Journal of Financial Mathematics, Vol. 3, Issue 1, , 2012, pp. 572-592.*

[8] *A. J. Lindeman, "Opportunities for Shared memory Parallelism in Financial Modelling," Proceedings of the IEEE Workshop on High Performance Computational Finance, 2010.*

[9] *K. Smimou and R. K. Thulasiram, " A Simple Parallel Algorithm for Large-Scale Portfolio Problems," Journal of Risk Finance, Vol. 11, Issue 5, 2010, pp.481-495.*

[10] *A. Srinivasan, "Parallel and Distributed Computing Issues in Pricing Financial Derivatives through Quasi Monte Carlo," Proceedings of the International Parallel and Distributed Processing Symposium, 2001.*

[11] *K. Huang and R. K. Thulasiram, "Parallel Algorithm for Pricing American Asian Options with Multi-Dimensional Assets," Proceedings of the 19th International Symposium on High Performance Computing Symposium, 2005, pp. 177-185.*

[12] *C. Bekas, A. Curioni and I. Fedulova, "Low Cost High Performance Uncertainty Quantification," Proceedings of Workshop on High Performance Computational Finance, 2009.*

[13] *D. Daly, K. D. Ryu and J. E. Moreira, "Multi-variate Finance Kernels in the Blue Gene Supercomputer," Proceedings of Workshop on High Performance Computational Finance, 2008.*

[14] *M. Majmudar, C. Docan, M. Parashar, C. Marty, "Cost vs. Performance of VaR on Accelerator Platforms," Proceedings of Workshop on High Performance Computational Finance, 2009.*

[15] *V. Agarwal, L. -K. Liu and D. A. Bader, "Financial Modelling on*

the Cell Broadband Engine," Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, 2008.

[16] C. Docan, M. Parashar, and C. Marty, "Advanced Risk Analytics on the Cell Broadband Engine," Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium, 2009.

[17] N. Singla, M. Hall, B. Shands and R. D. Chamberlain, "Financial Monte Carlo Simulation on Architecturally Diverse Systems," Proceedings of Workshop on High Performance Computational Finance, 2008.

[18] D. B. Thomas, "Acceleration of Financial Monte-Carlo Simulations using FPGAs," Proceedings of the IEEE Workshop on High Performance Computational Finance, 2010.

[19] B. Zhang and C. W. Oosterlee, "Option Pricing with COS Method on Graphics Processing Units," Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium, 2009. Simulation," Concurrency and Computation: Practice and Experience, 2012.

[20] D. M. Dang, C. C. Christara and K. R. Jackson, "An Efficient GPU-Based Parallel Algorithm for Pricing Multi-Asset American Options," Concurrency and Computation: Practice and Experience, Vol. 24, No. 8, 2012, pp. 849-866.

[21] K. H. Tsoi and W. Luk, "Axel: A Heterogeneous Cluster with FPGAs and GPUs," Proceedings of the 18th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 2010, pp. 115-124.

[22] M. Showerman, J. Enos, A. Pant, V. Kindratenko, C. Steffen, R. Pennington and W. -M. Hwu, "QP: A Heterogeneous Multi-Accelerator Cluster," Proceedings of the 10th LCI International Conference on High-Performance Clustered Computing, 2009.

[23] M. Byrne, F. Dehne, G. Hickey, and A. Rau-Chaplin, "Parallel Catastrophe Modelling on a Cell B.E.," Journal of Parallel, Emergent and Distributed Systems, Vol. 25, No. 5, 2010, pp. 401-410.

[24] O.A.C. Cortes, A. Rau-Chaplin, D. Wilson, I. Cook, and J. Gaiser-Porter, "Efficient Optimization of Reinsurance Contracts using Discretized PBIL," International Conference on Data Analytics, 2013, pp. 18-24.

[25] S. Salcedo-Sanz, L. O. Carro-Calvo, M. Claramunt, A. Castaner, and M. Marmol, "Effectively Tackling Reinsurance Problems by Using Evolutionary and Swarm Intelligence Algorithms", Risks, Vol. 2, No. 2, 2014, pp. 132-145.

[26] A. Rau-Chaplin and B. Varghese, "Accounting for Secondary Uncertainty: Efficient Computation of Portfolio Risk Measures on Multi and Many Core Architectures," Proceedings of the 6th Workshop on High Performance Computational Finance, No. 3, 2013, pp. 1-10.

[27] A. K. Bahl, O. Baltzer, A. Rau-Chaplin, and B. Varghese, "Parallel Simulations for Analysing Portfolios of Catastrophic Event Risk," Workshop on High Performance Computational Finance in the Proceedings of the International Conference of High Performance Computing, Networking, Storage and Analysis, 2012, pp. 1176-1184.

[28] A. Rau-Chaplin, B. Varghese, D. Wilson, Z. Yao, and N. Zeh, "QuPARA: Query-Driven Large-Scale Portfolio Aggregate Risk Analysis on MapReduce," Proceedings of the IEEE International Conference on Big Data, 2013, pp. 703-709.

[29] NVIDIA CUDA Compiler (nvcc) Version 5.0: https://developer.nvidia.com/cuda-toolkit-50-archive [Last accessed: 24 August 2015]

[30] Intel(R) Math Kernel Library Reference Library, MKL 11.0 Update 3, Document No. 630813-058US.

[31] CUDA Math Library: http://docs.nvidia.com/cuda/cuda-math-api/index.html [Last accessed: 24 August 2015]

[32] Boost Statistical Library: http://www.boost.org/ [Last accessed: 24 August 2015]

[33] Boost Beta Inverse Function: http://www.boost.org/doc/libs/1_59_0/libs/math/doc/html/math_toolkit/sf_beta/ibeta_inv_function.html [Last accessed: 24 August 2015]

[34] H. Posten, "An Effective Algorithm for the Noncentral Beta Distribution Function," The American Statistician, Vol. 47, No. 2, 1993, pp. 129-131.